Making Robust Networks using Adversarial Training

Christophe Ly - Guillaume Grosjean - Yoann Lemesle

December 21, 2021

Contents

1 Adversarial Attacks (White box)

Deep neural networks have been shown to be sensible to adversarial attacks, slight modifications of normal inputs that are able to fool models [3]. These modified inputs are called adversarial examples and the process of crafting them is called an adversarial attack. In this project we implemented some of the most known gradient-based (white box) attacks, the FGSM (Fast Gradient Sign Method) [3] and PGD (Projected Gradient Descent) [2] attacks.

The FGSM attack (see eq.1) simply consists in taking a single step of fixed size ϵ in the direction of the loss function's gradient with respect to the pixels of the images, thus increasing the value of the loss $l(x, y)$ by modifying each pixel's value by a small amount.

$$x' = x + \epsilon * sign(\Delta_x l(x, y)) \tag{1}$$

The PGD attack (see eq.2) is an iterative attack that consists in finding an adversarial perturbation inside of an l_p-norm ball of radius ϵ, taking a step of size η at each iteration.

$$x_{t+1} = \Pi_{B(0,\epsilon)}(x_t + \Pi_{B(0,\eta)}(\Delta_x l(x, y))) \tag{2}$$

The goal of this project was to try several defense mechanisms against these attacks, as well as try additional black box attacks.

2 Adversarial Defense Mechanisms

2.1 Adversarial Training

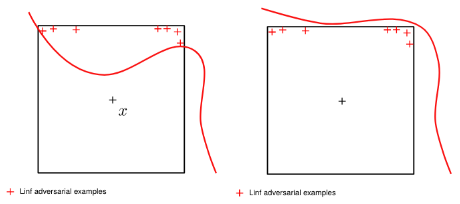

Figure 1: Illustration of the effect of adversarial training.

Adversarial training [2] is most of the most well known defenses against adversarial attacks. It simply consists of training models on adversarial examples instead of normal inputs (see eq.3). The intuition behind this idea is to "push" the decision boundary of the right class around adversarial examples in the hope that most of the inputs in the l_p-norm ball around natural examples are correctly classified 1. The radius and p-norm of this robust space then depend on the attack that is used during training (usually l_2 or l_{inf} PGD).

$$\min_{\theta} \mathbb{E}_{(x,y)} \left(\max_{||\tau|| \leq \epsilon} L_{\theta}(x + \tau, y) \right) \tag{3}$$

We implemented this method to train a small convolutional network (see table 1) on the CIFAR-10 dataset. We used the l_{inf}-PGD attack during training, using the parameters displayed in table 2. The robustness of the resulting model is compared with the robustness of a classical model in fig.2 using FGSM (parameter ϵ) and PGD (parameter ϵ, stepsize $\epsilon/10$ and 10 iterations).

Optimizer	SGD
Momentum	1e-5
Epochs	100
Learning Rate	Cosine Annealing (0.1 -> 0)
Data Augmentation	Random H Flips + Crops

Table 1: Training Hyperparameters

Attack	PGD l_{inf}
Iterations	10
Epsilon	10/255
Stepsize	2/255

Table 2: Adversarial Training Hyperparameters

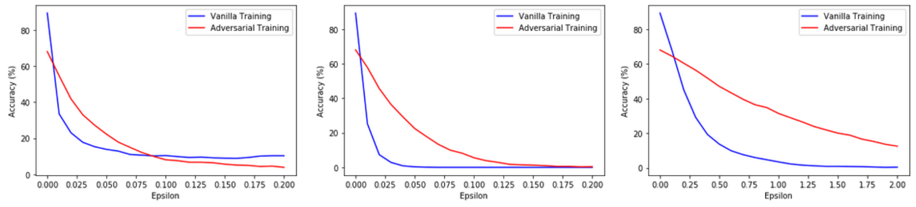

Figure 2: Robustness of the models with respect to the ϵ parameter of the FGSM (left), PGD-l_{inf} (center) and PGD-l_2 (right) attacks.

2.2 Gradient Norm Minimization

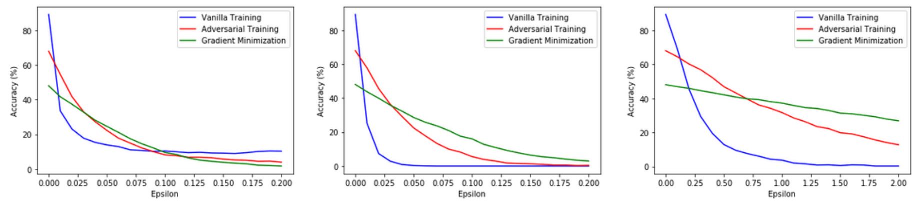

Figure 3: Robustness of the models with respect to the ϵ parameter of the FGSM (left), PGD-l_{inf} (center) and PGD-l_2 (right) attacks.

We wanted to try an intuitive idea that would be similar in spirit to adversarial training while being less heavy computationally. Instead of learning on adversarial examples to create a robust l_p-norm ball around natural examples, our idea was to "flatten" the curve of the loss function around natural examples in the hope that it would create such a "safe" space around input points. To do that, we added a simple constraint term to the loss function in order to minimize the l_2 norm of the loss gradients with respect to the inputs pixels (see eq.4).

$$L_{GNM}(x,y) = L(x,y) + \lambda * ||\Delta_x L(x,y)||_2 \tag{4}$$

Using the same training parameters as section 2.1, we obtained a network that, despite its low natural accuracy, seems to be more robust than the previous models (see fig.**??**). However, this method is an instance of **gradient obfuscation** : methods that makes the gradients noisy/small in order to confuse gradient-based attacks. These methods were shown to be easily bypassed **??**.

2.3 Randomized Networks

Among the popular defense mecanisms, we can find the randomization which has proven a pretty good efficiency in some context. The principles are simple, it consists in a noise injection both at inference and training time at selected layers. The common noises injected are drawn from the Exponential family such as Gaussian or Laplace distributions. For our experiments, we have chosen to use Gaussian noise, at first only during inference time, then during training. We got the following results for a basic classifier on CIFAR-10 with convolutional layers (73% on validation accuracy):

Attack	Accuracy without noise	Accuracy with noise ($\sigma = 0.25$)
FGSM ($\epsilon = 0.025$)	2%	15,8%
PGD L^∞ ($\epsilon = 0.01$, 10 iterations)	13%	33%
PGD L^∞ ($\epsilon = 0.025$, 10 iterations)	0.07%	16%

Table 3: Results on accuracy for noise injection at inference time for an attacked CIFAR-10 classifier.

For noise injection on training, we also managed to get similar results on accuracy under attacks. Overall, we can notice that the accuracy got better when we add randomization on our network. However, we can observe a slight drop for the natural accuracy going to 65% from 73% without noise injected. So, randomization appears as a good way to improve network robustness, but it still has to be used with an other defense technique such as the adversarial training we introduced.

2.4 Auto-encoders as adversarial defense

In the last assignment, we were introduced to auto-encoders and their uses for image denoising or anomaly detection. So, we thought of using these networks as a way to detect if an image is adversarial or not, and to clean it. This idea has already been explored by researchers with a network they called MagNet.

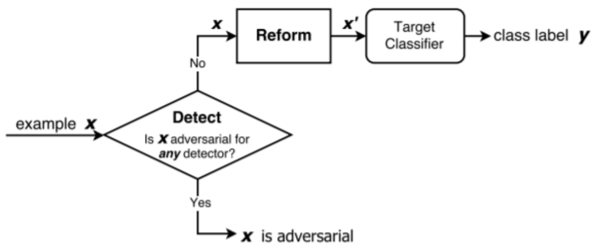

Figure 4: Workflow of the MagNet network.

Indeed, this network is using auto-encoders in two blocks:

- the "Detector": it uses an auto-encoder trained on normal examples, with the Mean Squared Error as loss function. It aims to approximate the manifold of natural examples with this auto-encoder. The Detector receives an image X as input, and passes it into its auto-encoder to check whether or not, this image is far from the manifold. To do that, it computes the reconstruction error between the output and the input of the auto-encoder. If the obtained value is above a predefined threshold t_re, we can not efficiently reconstruct the image. If not, we pass the image to the Reformer (see Fig. 4).
- the "Reformer": it uses an other auto-encoder trained on normal examples, with the Reconstruction error as loss function. It aims to push the adversarial examples to be close to the approximated manifold, by reconstructing them.

Then, the reconstructed images are passed into the original classifier. For our implementation, we only looked at the Reformer block due to a lack of time. It was also a good way to see why the Detector was needed, by trying to reconstruct images that were far from the manifold of normal examples. We trained an auto-encoder for the reformer for a short number of epochs (20) due to a lack of computational power, but it was still enough to get good results. We followed the architecture depicted on the MagNet paper.

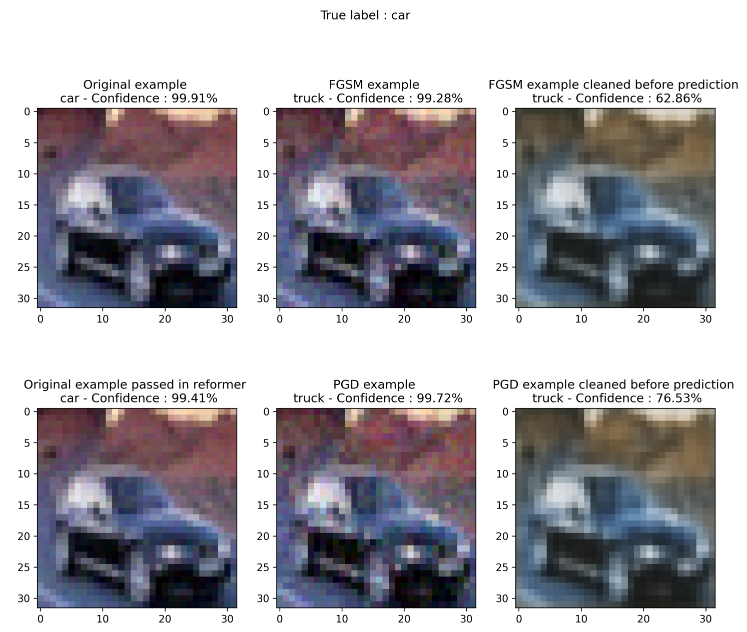
True label : car
Original example
car - Confidence : 99.91%
FGSM example
truck - Confidence : 99.28%
FGSM example cleaned before prediction
truck - Confidence : 62.86%
Original example passed in reformer
car - Confidence : 99.41%
PGD example
truck - Confidence : 99.72%
PGD example cleaned before prediction
truck - Confidence : 76.53%

Figure 5: Comparison between an attacked image and an reconstructed attacked image

Overall, we managed to have improvements on our predictions as we got the true label back as seen on the Fig. 5. For some pictures, even if we did have the good predictions, we still managed to highly reduce the confidence of the false labels. However, we also got worse results for some pictures that showed a terrible image reconstruction. This might be due to the short training of our auto-encoder or by the fact that we do not use a Detector. So, me may pass images that are too far from the manifold to provide good reconstructions. To have a better view on the efficiency of the MagNet, we can have a look at the results on accuracy:

Attack	Accuracy without AE	Accuracy with AE
No Attack	73%	67%
FGSM ($\epsilon = 0.01$)	22%	39%
FGSM ($\epsilon = 0.025$)	2%	17%
FGSM ($\epsilon = 0.05$)	0.4%	3.9%
PGD L^∞ ($\epsilon = 0.01$, 10 iterations)	13%	39%
PGD L^∞ ($\epsilon = 0.025$, 10 iterations)	0.07%	10%

Table 4: Results on accuracy for denoising with auto-encoder for an attacked CIFAR-10 classifier.

3 Black-box Adversarial Attacks

All attacks considered so far can be categorized as white-box attacks, i.e. we have access to the model parameters to make adversarial examples based on the gradient. In practical cases, it is more likely to only have access to the model predictions. Thus, we can't use backpropagation to compute the gradient : this is the black-box setting. Based on [1], we implemented black-box attacks in three different settings, going from least to most restrictive. All parameter used are available in table 5.

General	
σ for NES	0.001
n, size of each NES population	40
ϵ, l_∞ distance to the original image	8/255
η, learning rate	0.01
Partial-Information Attack	
ϵ_0, initial distance from source image	80/255
δ_ϵ, rate at which to decay ϵ	0.01
Label-Only Attack	
m, number of samples for proxy score	40
μ, ℓ_∞ radius of sampling ball	0.001

Table 5: Parameter used for black-box attacks.

3.1 Query-limited setting

In the query-limited setting, the attacker has a access to the prediction score for every label. The objective is to design a query-efficient algorithm : we want to make adversarial examples with limited number of queries. Let x_{true} be the image we want to attack, with label y_{true}. The idea is to find an adversarial image x_{adv} minimizing the true prediction $P(y_{true} \mid x)$. To do so, we compute an estimation $\nabla P(y_{true} \mid x)$ using NES algorithm [1]. NES is quite similar to finite-differences estimate but on a random Gaussian basis. Using this gradient estimation, we can run PGD to minimize $P(y_{true} \mid x)$ without ever using the model parameters. Example of a black-box attack in this setting is presented figure 6.

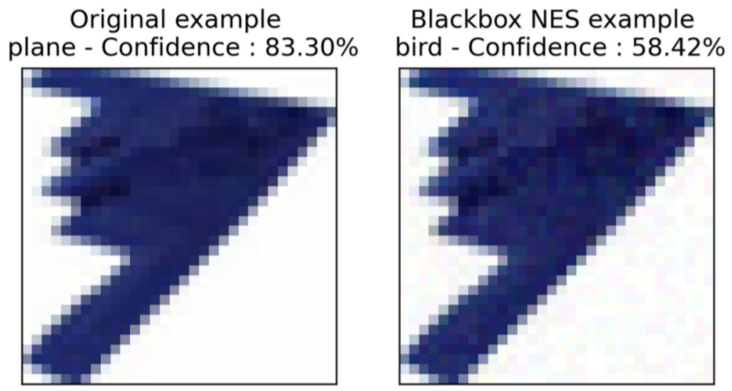

Figure 6: Adversarial example computed in the query-limited setting. Time for one attack : $\sim$ 1 second.

3.2 Partial-Information Setting

In the partial-information setting, the attacker only has access to the top k prediction of the model. We consider here the case $k = 1$, i.e. the attacker has access to the top prediction. In that case, we can't run the precedent algorithm because minimizing the true label prediction will make it disappear from the available data. We need to make a targeted attack.

Rather than beginning with the image x_{true}, we instead begin with an instance x_0 of the target class y_{adv}, so that y_{adv} will initially appear in the top class. Then, we alternate between projecting onto l_∞ boxes of decreasing sizes ε centered at the original image x_{true}, and maximizing $P(y_{adv} \mid x)$ in order to keep the target class prediction in the top prediction. While y_{adv} is in the top prediction, we can use NES to compute $\nabla P(y_{adv} \mid x)$ in order to maximize the adversarial prediction. We use the estimated gradient to run line search to find decreasing ε that maintains the adversarial class within the top prediction. Iterations of such an attack are plotted figure **??**.

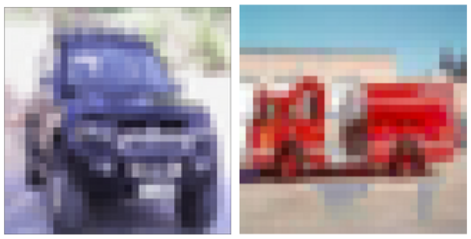

Figure 7: Left : original image x_{true} to be attacked. $y_{true} =$ Car. Right : starting image x_0. $y_a dv =$ Truck

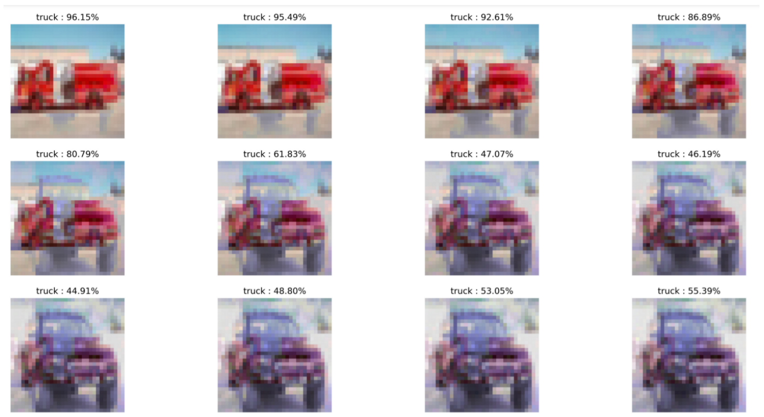

Figure 8: Partial-Information attack using images 9. Starting from an image of the adversarial class, decreasing epsilon make the image tend towards the original one, but always keeping the top prediction to be a Truck. Time for one attack : ~ 1 minute.

3.3 Label-Only Setting

In the label-only setting, the attacker only has access to the top prediction label, but not to the prediction score anymore. The idea is that, in absence of a prediction score, we need to compute an estimation of that value and then run the partial-information algorithm.

The paper introduce a discretized score of an adversarial example to quantify how adversarial the image is at each iteration simply based on the ranking of the adversarial label y_{adv}:

$$R(x) = k - \operatorname{rank}(y_{adv} \mid x) \tag{5}$$

that can be simplified if $k = 1$:

$$R(x) = \begin{cases} 0 & \text{if } y_{adv} \text{ is top prediction} \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

As a proxy for the softmax probability, the robustness of the adversarial image to random perturbations (uniformly chosen from a l_∞ ball of radius μ) is defined as :

$$\widehat{S}\left(x^{(t)}\right) = \frac{1}{n}\sum_{i=1}^{n} R\left(x^{(t)} + \mu\delta_i\right) \tag{7}$$

Using $\widehat{S}$ as a proxy for $P(y_{adv} \mid x)$, we run the partial-information algorithm to iteratively find an adversarial example. An example of a label-only attack is shown figure **??**.

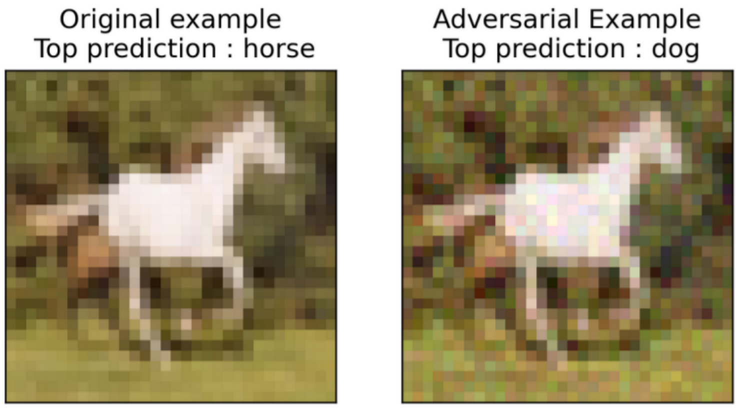

Figure 9: Adversarial example computed in the label-only setting. Time for one attack : $\sim$ 30 minutes.

References

[1] Andrew Ilyas et al. *Black-box Adversarial Attacks with Limited Queries and Information*. 2018. arXiv: `1804.08598 [cs.CV]`.

[2] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: `1706.06083 [stat.ML]`.

[3] Christian Szegedy et al. *Intriguing properties of neural networks*. 2014. arXiv: `1312.6199 [cs.CV]`.